Object-Oriented Programming
50:198:113 (Fall 2018)

Homework:	**1**	**Professor:**	**Suneeta Ramaswami**
Due Date:	**9/24/18**	**E-mail:**	`rsuneeta@camden.rutgers.edu`
Office:	**321 BSB**	**URL:**	`http://crab.rutgers.edu/~rsuneeta`
		Phone:	**(856)-225-6439**

Homework Assignment 1

This assignment will review material covered in the introductory course on Python. It requires you to write functions, use loops and lists, and process strings. In this and all future assignments, you are graded not only on the correctness of the code, but also on clarity and readability. I will deduct points for poor indentation, poor choice of object names, and lack of documentation. For documentation, use a common sense approach. While I do not expect every line of code to be explained, all code blocks that carry out a significant task should be documented *briefly* in clear English.

The assignment is due by 11:55PM of the due date (given above). The point value is indicated in square braces next to each problem. Each solution must be the student's own work. Assistance should be sought or accepted only from the course instructor or the TA. Any violation of this rule will be dealt with harshly.

Important note: When writing each of the following programs, it is important that you name the functions exactly as described because I will assume you are doing so when testing your programs. Points will be deducted if your program produces errors because the functions do not satisfy the stated prototype. In particular,

1. After importing the `problem1` module, I will call the `magic_list` function to test `problem1.py`.
2. After importing the `problem2` module, I will type `one_armed_bandit()` in the `Python` shell to test `problem2.py`.
3. After importing the `problem3` module, I will type `pig_latin_translator()` in the `Python` shell to test `problem3.py`.

For each problem, put all your function definitions in one file. *Do not create a separate file for each function.* The files `problem1.py`, `problem2.py`, and `problem3.py` should contain *all* the function definitions for Problems #1, #2, and #3, respectively. **Please read the submission guidelines at the end of this document before you start your work.**

Problem 1 [10 points] Magic numbers. A positive integer m greater than 1 is said to be a *magic number* if the sum of all its divisors (other than m) equals m. For example, 6 is a magic number because $1 + 2 + 3 = 6$, and 28 is a magic number because $1 + 2 + 4 + 7 + 14 = 28$, whereas 15 is *not* a magic number because $1 + 3 + 5 = 9 \neq 15$, and 24 is *not* a magic number because $1 + 2 + 3 + 4 + 6 + 8 + 12 = 36 \neq 24$.

In this program, you will write two functions, as described below. *Important note:* Do not repeat code unnecessarily. If you have a function to carry out a specific task, call that function rather than repeating those lines of code.

- *(5 points)* A function called `magic` with a single parameter n that returns `true` if n is a magic number and `false` otherwise.
- *(5 points)* A function called `magic_list` with a single parameter `num`. It **returns a list** of all magic numbers between 2 and `num` (inclusive). *Note:* There are not that many magic numbers out there!! To see a magic number other than 6 and 28, import your module in the Python shell and call `magic_list` with parameter 500. To see the next magic number, you will need to go even higher (over 8000).

Here is a sample output:

```
>>> magic_list(100)
[6, 28]
```

Problem 2 [20 points] One Armed Bandit. In casinos from Monte Carlo to Las Vegas, one of the most common gambling devices is the slot machine - the "one-armed bandit". A typical such machine has three horizontally aligned wheels that spin around behind a narrow window. Each wheel is marked with the following symbols: `CHERRY, LEMON, ORANGE, PLUM, BELL,` and `BAR`. The window, however, allows you to see only one symbol on each wheel at a time. For example, the window might show the configuration:

`BELL` `ORANGE` `BAR`

If you put a silver dollar into the slot machine and pull the handle on its side, the wheels spin around and eventually come to a rest in some new configuration. If that configuration matches one among a set of winning patterns printed on the front of the slot machine, you get back some money. If not, you're out a dollar. The following table shows a typical such set of winning patterns along with their associated payoffs:

`CHERRY`	–	–	pays	$2
`CHERRY`	`CHERRY`	–	pays	$5
`CHERRY`	`CHERRY`	`CHERRY`	pays	$7
`ORANGE`	`ORANGE`	`ORANGE/BAR`	pays	$10
`PLUM`	`PLUM`	`PLUM/BAR`	pays	$14
`BELL`	`BELL`	`BELL/BAR`	pays	$20
`BAR`	`BAR`	`BAR`	pays	$250

The notation `BELL/BAR` means that either a `BELL` or a `BAR` can appear in that position, and a dash means that any symbol at all can appear. Thus, getting a `CHERRY` in the first position is automatically good for two dollars, no matter what appears on the other wheels. Note that there is never any payoff for the `LEMON` symbol, even if you happen to line up three of them.

Write a program that simulates playing the slot machine. You will do this by writing a program with the functions described below.

- (*10 points*) Implement a function called `one_armed_bandit` without any parameters. This function should provide the player with an initial stake of $50 and then let her play repeatedly until the money runs out or the player decides to quit (do this with a `while` loop). During each round, your program should take away a dollar (since each spin of the wheel costs $1). Then spin the wheel and calculate the resulting payoff, if any (via a

function `spin_wheel`, described below). Then update the player's current money stash appropriately.

- (*10 points*) Implement a function called `spin_wheel` without any parameters to simulate the random spinning of the three wheels, print the result of the spins, and determine the resulting payoff. Your function should return the payoff amount. *Note:* You should assume that each of the six symbols is equally likely on each wheel: *i.e.* the function `spin_wheel` should mimic three rolls of a die. You can do this by using the `randint` function from the `random` module. We will discuss this function in class. (Slot machines in real life are usually biased towards the non-winning combinations to make things profitable for the casino in the long run.)

Here is an example of a (lucky) sample run:

```
>>> from problem2 import *
>>> one_armed_bandit()

                  The One-Armed Bandit Welcomes You
                  ---------------------------------

You have $50. Would you like to play (y/n)? y
PLUM     LEMON     LEMON  -- You lose

You have $49. Would you like to play (y/n)? y
PLUM     BAR       LEMON  -- You lose

You have $48. Would you like to play (y/n)? y
BELL     LEMON     ORANGE -- You lose

You have $47. Would you like to play (y/n)? y
CHERRY   CHERRY    ORANGE -- You win $5

You have $51. Would you like to play (y/n)? y
LEMON    ORANGE    BAR    -- You lose

You have $50. Would you like to play (y/n)? y
BELL     BELL      BELL   -- You win $20

You have $69. Would you like to play (y/n)? y
CHERRY   PLUM      LEMON  -- You win $2

You have $70. Would you like to play (y/n)? y
BAR      BAR       BAR    -- You win $250

You have $319. Would you like to play (y/n)? n
Goodbye! Come play again!
```

Problem 3 [20 points] Pig Latin. Pig Latin is a form of coded English often used for amusement. There are many variations on the method used to form pig Latin sentences. Here, we use the following simple algorithm to translate an English word into pig Latin:

- If the English word begins with a consonant, place the initial cluster of consonant letters at the end of the word, followed by the letters "`ay`" to it. For example, "`road`" translates to "`oadray`", "`scream`" translates to "`eamscray`", and "`fly`" translates to "`flyay`".

- If the English word begins with a vowel (a vowel is one of 'a','e','i','o', or 'u'), simply add the letters "`way`" at the end of the word. For example, "`apple`" translates to "`appleway`" and "`I`" translates to "`Iway`".

Write a program that translates English language sentences into pig Latin. We will assume that the sentence is made up of words separated by blanks. A word is a string of letters without blanks and may end in a punctuation mark, which we assume to be one of . (period) ! (exclamation) ? (question mark) or , (comma). We assume there are no other punctuation marks in the sentence. The punctuation marks and blanks stay as they are in the pig Latin translation. Implement the following functions in your program.

- *(8 points)* Write a function called `pig_latin_word` with a single parameter `word`. Assume that `word` is a string of characters without any spaces or punctuation marks in it. The function returns a string that is the pig Latin translation of `word`.
- *(7 points)* Write a function called `pig_latin_sentence` with a single parameter `eng_sentence`, which is an English language sentence. The function returns a string which is the pig Latin translation of `eng_sentence`. *Hint:* Construct the pig Latin translation by breaking up `eng_sentence` into words, each of which can be translated using `pig_latin_word`. Make sure you handle punctuation marks properly.
- *(5 points)* Write a function called `pig_latin_translator` without any parameters. The function repeatedly reads in an English sentence from the user and then uses the function `pig_latin_sentence` to print out the pig Latin translation of that sentence.

Here is a sample run:

```
                --------------------------------
                English to Pig Latin Translator
                --------------------------------

Enter the English sentence: Look! His tie is ugly.

In Pig Latin: ookLay! isHay ietay isway uglyway.

Do another? [y/n] y

Enter the English sentence: Are you speaking Pig Latin?

In Pig Latin: Areway ouyay eakingspay igPay atinLay?

Do another? [y/n] n

Goodbye!
```

Submission Guidelines

Implement the first problem in a file called `problem1.py`, the second one in a file called `problem2.py`, and the third one in a file called `problem3.py`. **Your name and RUID should appear as a comment at the very top of each file.**

Test each of your programs thoroughly before submitting your homework. When you are ready to submit, upload your files on Sakai as follows:

1. Use your web browser to go to the website `https:sakai.rutgers.edu`.
2. Log in by using your Rutgers login id and password, and click on the `OBJECT-ORIENTED PROG F18` tab.
3. Click on the 'Assignments' link on the left and go to 'Homework Assignment #1' to find the homework file (`hw1.pdf`).
4. Use this same link to upload your homework files (`problem1.py`, `problem2.py`, and `problem3.py`) when you are ready to submit.

You must submit your assignment at or before 11:55PM on September 24, 2018.